The Integration of LwM2M and OPC UA: An Interoperability Approach for Industrial IoT

Abdulkadir Karaagac, Niels Verbeeck, Jeroen Hoebeke
Ghent University - imec, IDLab, Department of Information Technology
Technologiepark Zwijnaarde 15, B-9052 Ghent, Belgium
{abdulkadir.karaagac, jeroen.hoebeke}@ugent.be

***Abstract*—Over the past years, Internet of Things (IoT) has been emerging with connected and smart things that can communicate with each other and exchange information. Similarly, with the emergence of Industry 4.0, the industrial world is also undergoing a strong evolution by connecting devices, sensors and machines to the Internet. In this paper, we investigate the integration of these two domains and examine the interconnection of two of the promising interoperability standards in these domains, namely OPC Unified Architecture and Lightweight Machine-to-Machine (LwM2M) protocol. For this purpose, we introduce an efficient and scalable approach, based on Docker Containers, for the cross-domain integration and interoperation. Besides, we also demonstrate and validate our interoperability approach by means of real world implementations and also theoretical and practical analysis.**

***Index Terms*—Industry 4.0, IoT, interoperability, OPC UA, Lightweight M2M, IPSO, docker containers**

I. Introduction

By means of the recent advancements in the Internet of Things (IoT) technologies, everyday, more and more things and objects are becoming smart and connected and smarter services are becoming reality, such as smart home, building and city [1]. Similarly, we also have seen the uptake of connected solutions in Industry to support and improve the operational performance in manufacturing, warehousing and distribution facilities. Especially, with the emergence of "Industry 4.0" [2], the Industry is rapidly evolving into a more automated and connected ecosystem with several applications, such as self driving cars, Automated Guided Vehicles (AGVs), and automatically controlled production lines.

Despite this plethora of technologies and the remarkable interest in the Industry 4.0 and IoT, their interoperation with and their integration into other ecosystems is still an open issue that requires innovation. Recently, we see several standardization initiatives and research efforts targeting interoperability and Machine-to-Machine (M2M) understandability in the IoT and Industrial domain. For instance, oneM2M [3], the Open Mobile Alliance (OMA) [4] and the IPSO Alliance [5] are some of the leading global organizations that deliver specifications and architectures for creating resource-efficient M2M communication and global interoperability for the IoT. On the other side, the Open Platform Communications Unified Architecture (OPC UA) [6] is a recent industry standard with several capabilities targeting machine to machine communication and interoperability for industrial applications. These efforts aim to create better interoperable and connective devices via common interfaces and data models. However, the devices and systems from these two worlds are still not able to communicate or interoperate with each other.

We believe that seamless and spontaneous interoperation of Industrial and IoT technologies would be extremely beneficial for both domains and would enable numerous new IoT and Industrial applications with more capabilities. Therefore, in this work, we study the first design for the integration of OPC UA [6] and LwM2M [7] protocols in the pursuit of the interoperability for Industrial IoT. Initially, we study the theoretical aspects (data types, methods, data models) and challenges for the integration of these technologies. Then, we introduce a scalable and efficient integration approach, by means of Docker Containers [8], where OPC UA Servers can be virtualized as LwM2M Clients and LwM2M Clients can be virtualized as OPC UA Servers with their objects and resources. In addition, we also validate our approach by means of practical implementation and detailed performance evaluation.

The remainder of the paper is organized as follows: Section II provides technical background about the target technologies. Then, the challenges and proposed approach for the integration of LwM2M and OPC UA protocols are described in Section III, followed by a description of the practical implementation in Section IV. After that, an evaluation of the implemented concepts is presented in Section V and finally, section VI concludes the paper.

II. Background

A. OPC Unified Architecture

The OPC UA [6] is an industry standard developed by the OPC Foundation [9] with several capabilities targeting machine to machine communication and platform independent operation for industrial applications.

First of all, OPC UA is a client-server protocol used for industrial communication between multiple devices and it offers an abstraction that describes what kind of messages can be exchanged between a client and a server. As illustrated in Figure 1, OPC UA is built on existing transport and security protocols such as TCP/IP, TLS and HTTP. An OPC UA server will provide data to clients through services as defined in [10]. Those services consist of a request-response structure similar to Service Oriented Architectures.

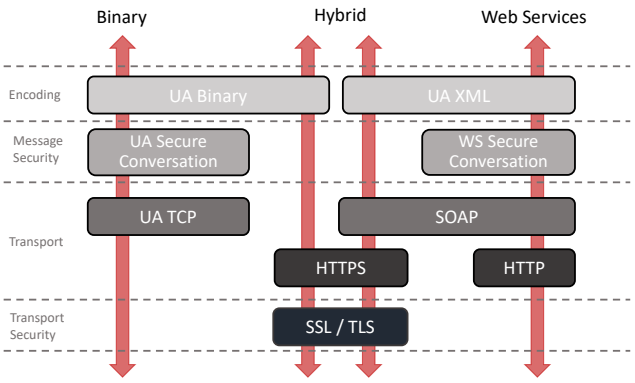

Fig. 1: OPC UA Stack

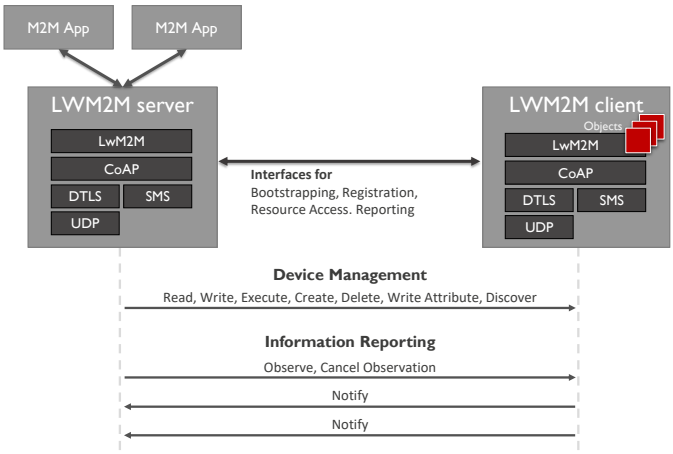

Fig. 2: LwM2M Protocol Stack and Interaction Model

In addition to communication aspects, data modeling is another important principal of OPC UA. The OPC UA object models, which may be defined by standardization organizations, vendors or end users, allow devices to provide type definitions for objects and their components, along with a globally unique identifier that can be used to provide description of the information meaning and semantics. The data modeling principals (e.g. type hierarchies and inheritance) of OPC UA allow to generate simple, but also very complex, information models. This data, that can be accessed by services, is stored in the address space of an OPC UA server [11], while type definitions of objects and data types are stored in the *Types* section, and instances of the objects are stored in the *Objects* section. Each object, both types and instances, contains a Node ID which uniquely identifies the object within the address space.

B. OMA LwM2M

LwM2M [7] is a secure and efficient client-server protocol, specified by the OMA Alliance, with several functionalities for managing resource-constrained devices on a variety of networks. Besides fundamental management functionalities such as bootstrapping, registration and firmware updates, LwM2M also defines efficient interactions for remote application management and the transfer of service and application data [7]. For this purpose, LwM2M provides several interfaces built on top of the Constrained Application Protocol (CoAP) [14], which is a REST-based application protocol for constrained Internet devices. Figure 2 presents the LwM2M communication stack and the interaction models related to device management and information reporting.

LwM2M, like OPC UA, uses a client-server architecture where a client can send CoAP messages (GET, POST, PUT and DELETE) to the server to retrieve and update information. Unlike OPC UA, CoAP uses UDP as transport protocol instead of TCP. As a result of this, the protocol is transformed into an asynchronous system in which requests and responses may not follow each other immediately.

According to LwM2M, a client consists of one or more instances of objects, which are typed containers that define the semantic type of instances. Each object is a collection of mandatory and optional resources, which are atomic pieces of information that can be read, written or executed. These objects, instances and resources are mapped into the URI path hierarchy with integer identifiers and can be accessed via simple URIs in the form of $/ObjectID/InstanceID/ResourceID$ [7]. For instance, a device model number can be read via a GET request to the URI "/3/0/1".

C. Related Work

In literature, there are a number of research attempts to extend OPC UA protocol and improve connectivity capabilities by any means. In [12], Gruner et al. investigates RESTful support in OPC UA, while [13] examines the transfer of OPC UA messages over CoAP protocol. In addition to a proxy (message translation) approach, the authors in [13] also considers encapsulating the OPC UA message inside a CoAP message. However, these works only target connectivity between OPC UA devices and other networks but they do not consider full-blown interoperability and understandability between OPC UA and IoT technologies and/or do not provide practical analysis about the proposed solutions.

III. Integration of LwM2M and OPC UA

The seamless connectivity and spontaneous interoperability of OPC UA and LwM2M technologies would be extremely beneficial for both domains and would enable numerous new IoT and Industrial applications with more capabilities. However, for IoT Devices, which typically have strict energy and memory constraints, the support of OPC UA functionalities, communication profiles and services is not practical. And the support for CoAP or LwM2M (or any of the IoT platforms) in OPC UA devices is currently not in the scope of OPC UA protocol and there is no clear intention from the foundation to include this support. Therefore, in this paper, we propose a scalable and efficient approach for the integration of these two worlds which can also be applied on already existing OPC UA and LwM2M devices, or also any other IoT interoperability protocol.

On one hand, some common aspects of these two technologies ease this integration process. For instance, both of these technologies harness client-server architecture. Moreover, they both define uniform interfaces and services and enable to define information and data models in a uniform way, which ensures the M2M understandability. And, they also use unique identifiers for the addressability of the information.

On the other hand, the fundamental differences in their design make it complicated to integrate these two interoperability protocols. First of all, OPC UA is based on a Stateful Communication and compulsory secure channels and sessions that will expire if they are not renewed regularly and it defines inherently stateful services (e.g. QueryNext, BrowseNext). In contrast, LwM2M/CoAP is based on UDP and defines a stateless communication scheme which is crucial for constrained devices. Secondly, OPC UA supports relatively more complex data models (e.g. type inheritance, nested object structure) compared to LwM2M, which only uses static object models that are typed containers that define the semantic type of instances and cannot consist of other objects. Also, LwM2M defines globally standardized information and object models with semantics by means of LwM2M object registry. However, for the case of OPC UA, associations and vendors can define object models, but there is no single authority which guarantees a global uniformity of object models and their identifiers. Last but not least, OPC UA supports a wide variety of data types and methods, where LwM2M only defines eight data types and existing CoAP methods.

1) Architecture: The proposed interoperability system is based on a virtualization/integration server that virtualizes the OPC UA and LwM2M devices in the other network by means of Docker Containers. As it is represented in Figure 3, an OPC UA network with OPC UA servers and clients and a LwM2M network with LwM2M Servers and Clients can transparently be integrated. The integration server in this network will contain both OPC UA and LwM2M Clients and Servers that can deliver application or management data from one protocol into another. This approach will result in virtualized objects from other networks as they are actually present in the target network domain and they can be accessed directly.

2) Data Types and Methods: As aforementioned, OPC UA defines more number of data types and methods compared to LwM2M. Therefore, a subset of these data types and methods cannot be mapped into LwM2M. Table I illustrates the mapping for data types and methods that we performed between LwM2M and OPC UA protocols. As it is illustrated in Table I, we used the *None* data type in LwM2M to translate the unmappable data types from OPC UA and mapped the available data types into corresponding data types. The OPC UA methods which does not have any counterpart in LwM2M cannot be supported, so are not included in this integration.

TABLE I: Data Types & Methods

Data Types		Methods	
OPC UA	LwM2M	OPC UA	LWM2M
String	String	Read	Read
Int16/32/64	Integer	Write	Write
Float	Float	Call	Execute
Boolean	Boolean	CreateMonitoredItems	Observe
ByteString	Opaque	CreateSubscription	
DateTime	Time	DeleteMonitoredItems	Cancel Observe
*	None	DeleteSubscription	

3) Object Model: The object structure of a protocol indicates how objects are represented and accessed. Therefore, for interoperation of target technologies, their object models needs to be understandable to each other. In this work, we achieved this by using a configuration file in which Node IDs from OPC UA are mapped to LwM2M Objects/Resources. As an example, Figure 4 provides the mapping for a temperature sensor resources in LwM2M and OPC UA.

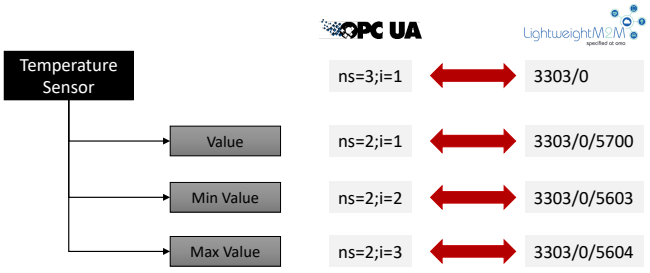

Fig. 4: Temperature sensor model in OPC UA and LwM2M

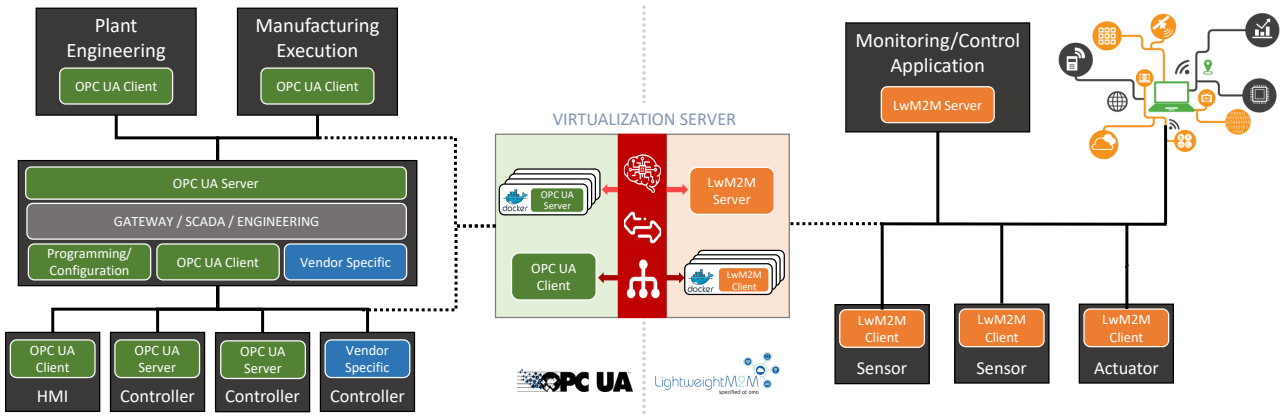

Fig. 3: Approach for the Integration of OPC UA and LwM2M Networks

For the mapping for the object models, we considered three options; Tag, Batch and IPSO Composite Object.

a) Tag Mapping: The *Tag approach* maps each OPC UA tag (Node ID) to a LwM2M Resource. This approach relies on static mapping of all of the Node IDs and corresponding LwM2M Resource URIs. It results in simpler configuration files, however it does not reflect the structures, relationships and/links between OPC UA tags or LwM2M resources into other domain.

b) Batch Mapping: This approach maps OPC UA objects, which may consist of variables and/or methods, to LwM2M objects by means of LwM2M Batch Object which was proposed in [17]. This batch object allows individual resources to be grouped, enabling them to be read and written together in a single request. In this way, multiple requests and their responses can be combined in in LwM2M and OPC UA Network. This leads to a reduction in network traffic.

c) IPSO Composite Object Mapping: Last mapping approach is based on *IPSO Composite Object* [16] and creates mapping between OPC UA objects and LwM2M Composite objects. In this way, OPC UA objects can be mapped to multiple grouped LwM2M objects. The difference with a batch object is that the configuration file specifies which OPC UA tag is an Input, Output or Setpoint tag and link it to a LwM2M resource with objlnk data type. Consequently, a LwM2M server can follow the link to reach the correct object.

IV. Implementation

In order to validate and evaluate the proposed approach, we developed an integration platform (runs a virtualization server) which connect the OPC UA devices into the LwM2M world. This virtualization server (implemented in Python) discovers OPC UA devices and resources and then expose these resources on self-managed LwM2M clients (named Anjay [18]) using a configuration file. The entire architecture consists of OPC servers, an OPC manager, Docker Containers, and a LwM2M server (named Leshan [19]).

As the schematic overview shows in Figure 5, the *OPC Manager* connects to the existing OPC UA servers and is responsible for starting up the corresponding Docker containers and forwarding particular traffic to the corresponding Container. Each docker container includes an adapter, a virtual device manager (VDM), and a stand-alone LwM2M client which includes all the resources defined in the configuration file for the corresponding OPC UA server. The *adapter* requests a configuration from the OPC manager after start-up and uses it to perform two tasks. On one hand, the *adapter* will send an observation request to the OPC manager with NodeIDs that it is interested. Next, the *adapter* will ask the VDM to create the corresponding LwM2M object instances and receive back their URIs. This will create a link between an OPC UA NodeID and an LwM2M URI in the *adapter*. So, any update for those NodeIDs in the OPC server will be sent to the corresponding LwM2M client and write commands from LwM2M client are also forwarded to the corresponding OPC server. So, any change on the OPC UA devices are reflected to the virtualized LwM2M clients, while any change applied to the virtual LwM2M client will be forwarded to the corresponding OPC UA device.

Regarding the object model mapping, we applied the three approaches defined in the previous section. An example configuration for mapping of a temperature sensor is provided in Figure 6. For the Tag mapping, each OPC UA tag resulted in a LwM2M resource. While, for the Batch, in addition to individual resources created for OPC UA tags, a Batch object instance is created which combines these individual resources. Lastly, the *IPSO Composite Object* is used by the adapter to map an OPC UA object to a LwM2M object. After the LwM2M objects are created individually on the LwM2M client based on the configuration file, the Input, Output and Setpoint link of the IPSO composite object [16] will be filled in with a reference to the individual objects.

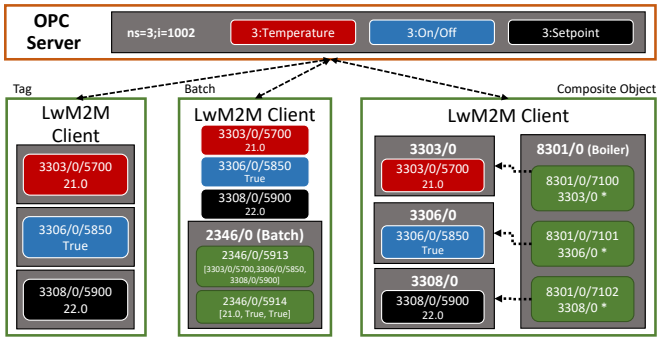

Fig. 6: Configurations for object mapping approaches

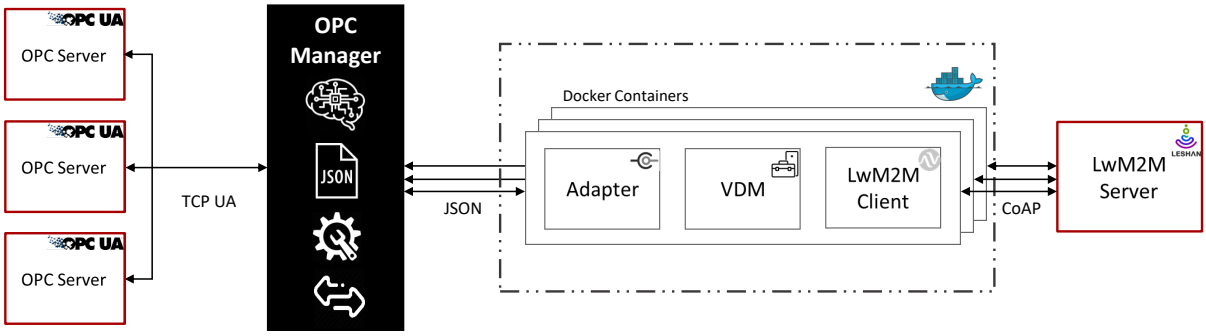

Fig. 5: Implementation Overview

V. Evaluation

In this section, we present the performance evaluation of our integration approach and implementation in terms of scalability, efficiency and latency. In order to perform this evaluation, we created the setup in Figure 7, on different server machines. A machine is used to run the OPC-servers and another one to run the OPC Manager and several docker containers and another one to run a LwM2M Server which interacts with virtualized LwM2M Clients.

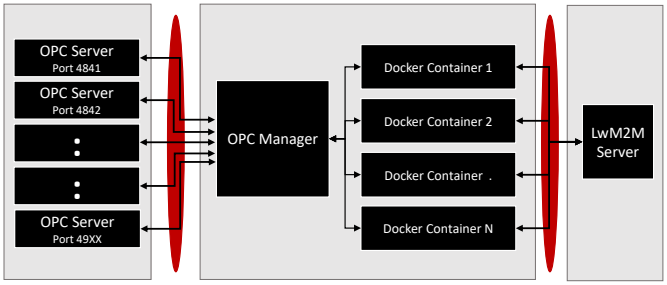

Fig. 7: Evaluation Setup

A. Scalability

Scalability test targets determining the maximum number of servers and tags (with certain update rates) that can be supported by the virtualization server. For this test, we first run a scenario with varying number of OPC UA servers which holds a single tag, then we performed the similar measurements by running single OPC UA server which holds varying number of OPC UA tags.

1) Multiple OPC UA Servers, Single Tag: In this scenario, we run a setup with varying number of OPC UA servers that hold a single Tag and we measured the maximum update request rate that can be handled for the given setup. As illustrated in Figure 8, a maximum of 160 requests/s can be achieved for 10 OPC UA servers. As the number of servers increases, the number of requests per second decreases, to the point where 150 servers can be supported at an update rate of 2.5 requests/s. Another outcome of this study is that the virtualization server can handle a total request rate of maximum 1600 requests/s, but this numbers starts to decrease after when the number of OPC UA servers exceeds a certain level. This is due to the fact that increasing number of virtualized OPC UA servers (consequently TCP connections) is affecting the performance of the OPC Manager.

2) Single OPC UA Server, Multiple Tags: In the second scenario, we run only a single OPC UA server which holds a varying number of Tags and we measured the supported update rates that can be handled by the Virtualization Server for the given number of Tags. As it is shown in Figure 9, the supported update frequency decreases as we increase the number of Tags exposed by the OPC UA server. In case of 10 Tags, an update rate of 32 requests/s is achieved, while only 2.5 requests/s update rate could be achieved when the OPC UA server exposes 280 Tags.

According to this figure, the curve for the total requests per seconds shows that a maximum of 700 updates could be

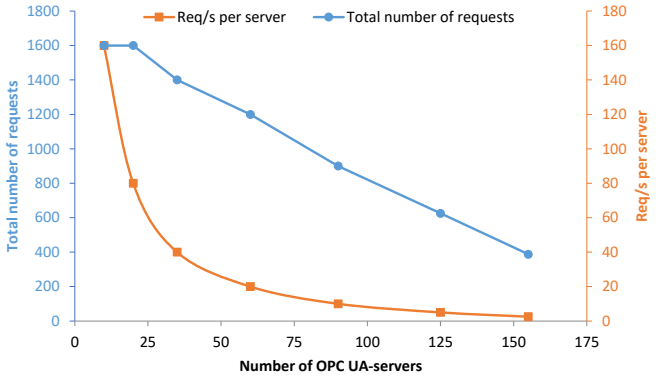

Fig. 8: Scalability: Multiple OPC UA Servers & Single Tag

Fig. 9: Scalability: Single OPC UA Server & Multiple Tags

forwarded to the LwM2M Server which is a lot lower than the maximum value of the previous scenario, 1600 requests/s. Even this number was not achieved when there were very low or high number of Tags. The performance drop for the high number of Tags was expected due to increased load on the virtualization server. But, the reason for the performance loss in the low values was the fact that the OPC UA server was experiencing limitations in updating the value of a certain Tag in real time. But in real OPC UA Servers running on Real Time Industrial Equipments, this issue would not appear.

B. Efficiency

The comparison for the data traffic in the OPC UA and LwM2M network between batch and without batch can be found in Table II.

First, the OPC UA network is observed, where messages are sent via the TCP protocol. Therefore, an OPC UA Request response will always consist of 4 messages (request, ACK, response, ACK). When comparing the update of one tag with the update of 10 tags, without using the batch implementation, it can be shown that updating 10 tags at the same time has a beneficial impact on network traffic because the ACK messages are merged with the request response messages. A total of 22 messages will be required for 10 tags, whereas, for one message, 4 messages will be required. By bundling 10 write requests, the number of bytes per tag can be reduced by a quarter.

TABLE II: Efficiency Analysis

	# of packets (Req/Resp/ACK)	# of bytes (Req+ACK)	# of bytes (Resp+ACK)	Total Bytes	Bytes /value
No Batch - OPC UA					
1 Tag	2+2	163+68	132+68	431	431
10 Tags	2*10+2	163*10+68	132*10+68	3086	308.6
No Batch - CoAP					
1 Resource	2	71	56	127	127
10 Resources	2*10	71*10	56*10	1270	127
Batch - OPC UA					
1 Tag	2 + 2	424+68	168+68	728	728
10 Tags	2 + 2	424+68	168+68	728	72.8
Batch - CoAP					
1 Resource	2	106	56	162	162
10 Resources	2	106	56	162	16.2

Whereas, in CoAP network, a write communication only consists of 2 messages. Again, a comparison can be made between updating one versus 10 resources, without using a batch implementation. By grouping the resources in a batch object, the total number of bytes is reduced by a factor of 8. The disadvantage of modifying one resource that is part of a batch object is much more limited with CoAP than with OPC UA. Here, the number of bytes per resource is only a factor of 1.3 greater than sending a normal update message.

These measurements show that the batch operation can improve the bandwidth efficiency in the both of the networks drastically. However, the disadvantage is that when only one tag is changed, the nine other tags are also transmitted, which doubles the number of bytes per tag.

C. Latency

The last test examines the time responsiveness of our implementation by measuring the delay between the generation of a write request from LwM2M server and its delivery to the corresponding OPC UA server. When the test was performed, 10 snapshots were recorded in *Wireshark*, consisting of 5 batch and 5 ordinary write requests. In each case, the transmission time of the first CoAP packet and the reception time of the last OPC UA packet was noted. The time (t) between the two determines the time needed to transfer a write request from CoAP to the OPC UA network. When comparing the averages in Table III, it becomes clear that sending a *batch* operation is clearly faster than sending 10 stand-alone requests.

TABLE III: Latency Analysis

	First CoAP message timestamp (s)	Last OPC UA message timestamp (s)	Δt(s)
No Batch			
RUN 1	57.90930	58.07579	0.166491
RUN 2	39.45434	39.61770	0.163365
RUN 3	90.14138	90.31221	0.170827
RUN 4	39.20719	39.36196	0.154773
RUN 5	67.59415	67.76161	0.167453
Average			0.164582
Batch			
RUN 1	12.32476	12.38791	0.063158
RUN 2	14.25634	14.29041	0.034067
RUN 3	12.97906	13.02896	0.049901
RUN 4	11.92192	11.95757	0.035649
RUN 5	13.32792	13.35941	0.031487
Average			0.042853

VI. Conclusion

In this work, we investigated the integration of OPC UA and LwM2M protocols in the pursuit of the interoperability for Industrial IoT. We introduce a scalable and efficient integration approach, by means of Docker Containers [8], where OPC UA and LwM2M devices are virtualized in the contrary network with their objects and resources. Besides the theoretical study, we also demonstrate and validate our approach by means of practical implementation and detailed performance evaluation.

We also performed experimental evaluations for our approach and implementation in terms of the scalability, latency and throughput. These evaluations showed that our implementation is able to support 10 OPC UA servers with an update rate of 160 update/sec, which means handling more than 1600 requests in total. Moreover, we also studied the contribution of the *Batch* object in the Network performance in terms of latency and throughput. Regarding the performance of the Batch, updating 10 resources in a batch is 4 times faster than updating the individual resources. In addition, the number of bytes in OPC UA is reduced by a factor of 4.2 and by a factor of 7.8 in LwM2M.

References

[1] Li, S.; Xu, L.D.; Zhao S. The Internet of Things: A Survey. In *Information Systems Frontiers*; Springer: Berlin, Germany, 2015; v. 17, pp. 243-259.

[2] Digital transformation in the manufacturing industry: challenges and accelerators. Available online: https://www.i-scoop.eu/digital-transformation/digital-transformation-manufacturing (accessed on 10 July 2018).

[3] OneM2M. White Paper: The Interoperability Enabler for the Entire M2M and IoT Ecosystem. 2015. Available online: http://www.onem2m.org/images/files/onem2m-whitepaper-january-2015.pdf, (accessed on 22 May 2018).

[4] Open Mobile Alliance. Available online: http://openmobilealliance.org (accessed on 22 May 2018).

[5] IPSO Alliance, Internet Protocol for Smart Objects (IPSO). Available online: https://www.ipso-alliance.org (accessed on 22 May 2018).

[6] OPC Foundation, *OPC Unified Architecture Specification Part 1: Overview and Concepts*

[7] Open Mobile Alliance. *Lightweight Machine to Machine Technical Specification, Approved Version 1.0*; 2017.

[8] Docker: Software Containerization Platform. Available online: https://www.docker.com, (accessed on 10 July 2018).

[9] OPC Foundation: The Industrial Interoperability Standard. Available online: https://opcfoundation.org/ (accessed on 10 July 2018).

[10] OPC Foundation, *OPC UA Specification: Part 4 Services*

[11] OPC Foundation, *OPC UA Specification: Part 3 Address Space Model*

[12] S. Gruner, J. Pfrommer, F. Palm, *"RESTful Industrial Communication With OPC UA,"* in IEEE Transactions on Industrial Informatics, vol. 12, no. 5, pp. 1832-1841, Oct. 2016.

[13] P. Wang, C. Pu, H. Wang, Y. Yang, L. Shao, J. Hou, *"OPC UA Message Transmission Method over CoAP", March 2018.*

[14] P. van der Stok, C. Bormann, A. Sehgal, *"PATCH and FETCH Methods for the Constrained Application Protocol (CoAP),"* April 2017. https://tools.ietf.org/html/rfc8132.

[15] M. Van Eeghem, *"Extensions to LwM2M for improved efficiency and interoperability",* Master's thesis, Universiteit Ghent 2017.

[16] J. Jimenez, M. Koster, H. Tschofenig, *IPSO Smart Objects* http://www.ipso-alliance.org/wp-content/uploads/2016/01/ipso-paper.pdf.

[17] A. Karaağaç, F. Van Den Abeele, J. Hoebeke, *"Challenges for semantic LwM2M interoperability in complex IoT systems",* WISHI: Workshop on IoT Semantic/Hypermedia Interoperability, 2017.

[18] AVSystem, *Anjay: open-source LwM2M Library* https://www.avsystem.com/products/anjay, accessed on 1 July, 2018.

[19] Eclipse, *Leshan: An OMA Lightweight M2M (LWM2M) implementation in Java* https://eclipse.org/leshan/, accessed on 1 July, 2018.